AFRL-RI-RS-TR-2012-258

TRANSFORMATIVE RENDERING OF INTERNET RESOURCES

OCTOBER 2012

INTERIM TECHNICAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2012-258 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /

JAMES PERRETTA
Chief, Cyber Assurance Branch

/ S /

WARREN H. DEBANY, JR., Technical Advisor
Information Exploitation & Operations Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* **OMB No. 0704-0188**

1. REPORT DATE *(DD-MM-YYYY)* OCTOBER 2012	**2. REPORT TYPE** INTERIM TECHNICAL REPORT	**3. DATES COVERED** *(From - To)* SEP 2009 – SEP 2012

4. TITLE AND SUBTITLE TRANSFORMATIVE RENDERING OF INTERNET RESOURCES	**5a. CONTRACT NUMBER** IN HOUSE
	5b. GRANT NUMBER N/A
	5c. PROGRAM ELEMENT NUMBER 62788F
6. AUTHOR(S) Frank H. Born	**5d. PROJECT NUMBER** GAIH
	5e. TASK NUMBER CY
	5f. WORK UNIT NUMBER BR
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/Information Directorate, Rome Research Site/RIGA, 525 Brooks Road, Rome NY 13441-4505	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/Information Directorate, Rome Research Site/RIGA, 525 Brooks Road, Rome NY 13441-4505	**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2012-258

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA# 88ABW-2012-5089
Date Cleared: 24 Sept 2012

13. SUPPLEMENTARY NOTES

14. ABSTRACT
There needs to be a reliable way to protect the client browser from any malicious code that is hosted on many websites. The technology chronicled in this report is called “Remote Web Rendering”. It takes the approach to remotely render the Web page such that it can be completely re-written before it gets to the browser. The re-written code will only pass on known good code to the browser. Remote Web Rendering is a simple but profound concept that is capable of completely insulating users from both known and unknown web based threats. The power of Remote Web Rendering is that it does not rely on code analysis methods to detect malicious code. It implements protection against all incoming code regardless of whether signature analysis methods can detect if it is malicious.

15. SUBJECT TERMS
Browser, Malware, Web Application, Filter, Signature Analysis, Remote Rendering

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**E. PAUL RATAZZI**
U	U	U	U	31	**19b. TELEPONE NUMBER (*Include area code)*** **N/A**

Table of Contents

Executive Summary

Malicious code is surreptitiously attached by hackers to many legitimate sites with the sole purpose to silently infect the computer that accesses the infected page. Even security vendor's web sites have fallen victim to these attacks[i] and, in turn, attacked the visitors to their site[ii]. Many other security vendor sites have been shown to be vulnerable also[iii,iv,v]. Given that we have no control over the content of the pages that we visit how can we protect ourselves from the ever present threat? Malware signature based protection schemes will only protect us from a subset of the known (non-mutated) threats. Real time code analysis could detect some new threats but it is no match for the obfuscation techniques employed in most attacks. What we need is a solution that can protect us from known or unknown (or mutated) threats without having to first identify the threat.

Given that malware will be hosted on many legitimate pages for many years to come there needs to be a reliable way to protect the client browser from any malicious code that they have accessed. Disrupting malware that is resident on a Web page can be done in several ways. The predominant way is to filter the incoming page by looking for signatures of known malware. The problem with this technique is that it is dependent on signature analysis that is ineffective against new attacks or obfuscated attacks. A technology chronicled in this report is called "Remote Web Rendering". It takes the approach to remotely render the Web page such that it can be completely re-written before it gets to the browser. The re-written code will only pass on known good code to the browser. Remote Web Rendering is a simple but profound concept that is capable of completely insulating users from both known and unknown web based threats. The power of Remote Web Rendering is that it does not rely on code analysis methods to detect malicious code. It implements protection against all incoming code regardless of what up to date signature analysis methods might say about its safety.

Remote Web Rendering also has a side benefit. Since the page is rendered remotely there is no possibility for the web site to determine what particular client accessed their page (unless they are required to login). For clients who are concerned about industrial espionage, or, from a government viewpoint, Operations Security (OPSEC), this non-attribution greatly increases the privacy of their searches and page access.

Introduction

The Web carries a vast amount of malicious code and is the platform from which the majority of attacks against personal computers are launched. Viruses, Trojan horses and other malicious code accompany millions of web pages. Of these infected pages approximately 90% are legitimate pages that have been hacked[vi]. Only 10% of the infected pages were purposely bult to host malware. The Web browser provides an avenue for direct compromise of the user's machine. In many cases all it takes is for the user to navigate to an infected web site and the exploit will automatically launch and try to infect their machine.

Personal computers that are infected through web based malware are often under the control of attackers who can use them for personal gain, economic or even military purposes. The technique presented here could be useful for protecting government as well as private sector computers. It presents a low cost method for users to protect themselves from being infected and possibly unwittingly becoming an agent of the enemy. For home use it is possible that the cost of this technology could be funded entirely through alternate means such as advertisement revenue.

Remote Web Rendering is a simple but powerful concept. By remotely rendering the web page, and passing on a re-written page in an innocuous form, you can insulate the client web browser from all malicious code on the original web page. The rendering server keeps the original code away from the client browser. Since the rendering server does not pass on any original code there is no need for analysis to determine what code may be harmful and what is benign. It is important to understand the method by which content is identified by the rendering server. The rendering server will render the web page using either the Firefox or Google Chrome rendering engine. The rendering server then captures a screen shot of the page and creates code that positions elements of the web page over the top of the screenshot. Content that is interactive or arrives at a later time will be displayed to the user in an alternate form that cannot infect their computer.

For the client, Remote Web Rendering will require no specialized hardware nor will it require any software installation. In some cases the user may have to set some configuration items in their browser such as setting up a proxy connection or lock down a few security settings. Since this technology is lightweight it should be useful for the un-patched masses (the common folk who do not keep their software patches up to date). In addition it could also be useful for the highly managed enterprise where the technology would be another layer in their defense systems.

The technology underpinning this technique is the subject of a patent application entitled "Transformative Rendering of Internet Resources"[vii]. This technology is available for licensing through the US Air Force by referencing patent pending application serial number 12/802,458, filed May 13, 2010.

Related Work

Other researchers have started from the same conclusion that the client can only be truly safe from internet based malware if their web content is rendered remotely. Where the research diverges is about how to send the client a safe version of the remotely rendered web resource.

- One approach for presenting a safe version of the page is through the use of a thin client approach. The Cross Fabric Internet Browsing System (CFIBS) was developed by Air Force Research Laboratory Sensors Directorate (AFRL/RY). CFIBS uses thin client hardware and a KVM switch (Keyboard, Visual, Mouse) to provide a separate secure channel for the client to browse the internet securely[viii].
    - **Comparison to our approach:** While CFIBS technology completely protects the client machine from compromise due to online malware it is not meant for large scale use since it requires dedicated hardware for each client. Our technology (Remote Web Rendering) described here does not require hardware or software to be installed for the client thus it could be used by the masses both within and external to large enterprises.
- Browser Shield uses “vulnerability driven filtering” and dynamic exploit removal to accomplish page sanitization for security purposes. It utilized interposition techniques in the browser to rewrite web based scripts into safe forms[ix].
    - **Comparison to our approach:** This technique relies on an understanding of known vulnerability types and wrapping of JavaScript. Remote Web Rendering does not rely on any knowledge of vulnerability types since this knowledge is always changing. In addition, the wrapping of JavaScript code is in a way similar to sandboxing but that sandbox is likely to be something that a clever programmer could escape from.
- Web Shield employs the same method of using a “middle box” to render the web pages and passes no untrusted JavaScript on to the client. Rather it runs the JavaScript code at the middle box and runs a JavaScript rendering agent at the client. The html is transferred to the client by transferring the encoded DOM of the web page[x].
    - **Comparison to our approach:** Differences between Web Shield and Remote Web Rendering revolve around the way in which the safe

code is transferred to the client. While RWR presents screenshots overlaid with HTML elements, Web Shield will send pure HTML code to the client whenever the DOM on the remote machine changes. While this technique should be more responsive than the screenshot method it will suffer on many pages due to the frequent changes that the DOM goes through.

Benefits

Many web sites are created just for the purpose of hosting malware. These largely consist of pornographic sites, gambling sites, or sites that capitalize on celebrity news or other hot search topics. Users are often educated to stay away from this type of site because of the likelihood that they could contain malware. The more insidious threat comes from legitimate web sites that have themselves been hacked. There is no way of anticipating which of these sites have been hacked and therefore pose a security threat to visitors. The purpose of most of this web page hacking is to plant malicious code on the web site that will attack any computer that accesses that web page. News sites, on-line stores, government sites even computer security sites have all played innocent host to the malicious code.

Consider how a targeted attack could be launched through the Web. By planting malware on an intranet site a whole enterprise could be targeted. Similarly, malware planted on a DOD contractor’s site would end up targeting the likely customers of that site - namely DOD agencies and their contractors.

In addition to web based malware there are other threats out there that this technique will foil. For example consider the following benefits for the remote Web rendering technology:

Malware removal

The main objective of the remote Web rendering technique is to ensure that no malicious code is transferred to the browser. Since the rendering server only passes on benign content to the client they are protected from all malware variants whether known previously, mutated or entirely new (zero day).

Securing WiFi Connections

Any unencrypted wireless connection is subject to listening by others “within range” of the signal. (Note that with even rudimentary antennas WiFi signals can be accessed from far away). Unencrypted Web applications that the user is accessing can easily be “sidejacked” while they remain open. Remote Web

Rendering can provide an SSL connection that encrypts traffic passing through a public wireless system. Sidejacking with an application such as Firesheep will no longer be an issue.

Disrupt Machine Fingerprinting

Fingerprinting of a computer has been gaining attention recently. It goes far beyond cookie based tagging. It allows web pages or advertizing inserts to identify a user based on the parameters of their machine and browser such as the fonts that are installed, the versions of plugins, and operating system and browser parameters. This can be done even if the user is security conscious and does not allow persistent cookies to be placed on his machine. Remote Web Rendering will break this fingerprinting process. Instead the tracker will only see parameters that apply to the rendering server. None of the client machine parameters will be accessible to the web site owner.

Break Steganographic Techniques

Capturing an image of an image breaks many (but not all) steganographic techniques. Capturing an image of a formatted text file can also break some text based covert channels.

Covert Channel Disruption

Since HTML disregards multiple whitespaces and treats them as a single whitespace it is possible to use this and other flexibility in HTML to encode data that is being passed to the client system. While this capability is a side benefit of the Remote Rendering technology it will probably only be useful in limited cases. For example, in a prison or detention center access to the internet might allow coded messages to be passed to the inmates.

Disrupting Exfiltration

Just as covert channels can be used to send data to a client they also can be used to exfiltrate data from the client system or web server. By running the remote web rendering system in a reverse proxy mode (re-writing the pages as they leave the server) it will break many of the covert channels that could be used to exfiltrate data off of the system.

Reverse Malware Blocking

In a situation where Malware is planted on an intranet site the target of the malware will be the employees of that company. If a military web site was hacked then the primary targets of the malware would be government employees and their contractors. Using Remote Web Rendering techniques to re-write the pages as they leave the server we will protect the clients for those pages.

There are lots of ways for the attacker to plant the malicious code on the site but there is only one way for the infected site to attack the client – that being direct download of the page and display in a browser.

Page Re-Writing Toolbox

The Web browser provides one of the most flexible platforms for software development. There are many ways to accomplish the same task. We have already mentioned some of the tools that we are using in re-writing web pages. Below is a list of the useful tools for this effort:

Tool	What it is	How we use it
Image Maps	HTML format that overlays items on top of an image	This is the basis for our re-written web pages. Content items will be overlaid on the background image.
DOM	Document Object Model - The representation of the structure and elements of the page.	The DOM catalogs the existence and location of the elements that are to be overlaid on the image map.
Layers	Web pages can utilize multiple visible or invisible layers.	We can use these layers to hold items such as dropdown menu choices, mouse-over images, form processing actions etc. They can be pushed to front or back using CSS/JavaScript controls.
HTML 5	New Web standard that contains advanced capabilities for content presentation.	Capabilities resident in the new HTML specification will allow conversion of vulnerable formats such as Flash into standard HTML.
AJAX	Asynchronous JavaScript and XML - Provides the ability to add content or	AJAX code is used to continually pass data in both directions, from the client to the Rendering Server (RS) and from the RS to the client.

	receive user input without reloading a page.	
Caching	Saved renderings of recent pages	If the rendering server is used by multiple people then the caching of recent pages will make the browsing experience much faster. It will also decrease the footprint of an agency as seen by search providers and others.
Code Compression	Creating a smaller file size version of the original web page	The re-written page code is significantly smaller than the original page code. Overall the code is very clean and concise in the new page and the display time when it reaches the client machine is fast.
Big Pipe	The rendering server will typically be hosted on a much faster connection than that available to the client.	The original page will arrive faster at the server than it would at the client. Since the re-written page will be smaller than the original page it will transfer to the client quicker than the original page.
Deconstruction	Separate vulnerable file formats into vulnerable and non-vulnerable parts.	Another way to deal with Flash elements (and similar) is to separate the content into elements that can safely be passed on or need to be re-created. Deconstruction allows dangerous elements to be transformed to another format.
File Format Converters	Convert one file format to another	File types that can contain malicious code can be converted to alternate file types that present little danger to the client. I.e. PDF documents are easily converted to images.
Auto Refresh	At a specified interval the page and/or screenshot can be	Auto Refresh can be used to push updated versions of pages, or portions of pages, at regular intervals.

	automatically refreshed	
Click-Through	Directly passing on mouse actions to the rendering server	In certain cases it may be necessary to pass mouse actions to the rendering server or directly to the original page source. In turn the source will provide response content through the rendering server.

Use Cases

Remote Web Rendering is not envisioned to be a complete replacement for the current client browsing experience. Certain aspects of the modern browsing experience would be hard to totally duplicate with code that is automatically generated by the rending server. While we do not envision this as a one size fits all solution, the utility of this concept is quite compelling in many cases where security is important. Here are a few examples:

Battle zone buffering

Soldiers on the battle zone obviously are key information targets for the enemy. On the other hand, internet access has almost become a human rights issue. This is a delicate balancing act to allow free internet surfing without jeopardizing the mission. Remote Web Rendering may be the technology that provides the insulation from threats that is absolutely required in this environment.

Critical computer buffering

Remote Web Rendering presents an additional safeguard for highly critical computers. Signature analysis can only work against known malware signatures. New (unknown) malware is what will be used against critical targets. Signature analysis cannot detect these attacks.

Casual surfing

Almost any computer that is used for business (or mission!) should be considered a target for either economic crime, espionage or trade secrets theft purposes. A single infected enterprise computer can often become the segue by through which an organization’s intellectual capital is stolen. Even if these computers are only protected during off duty time it could still result in a significant reduction in risk to the organization.

Home user safeguard

The un-patched masses are quite often the minions of the bot masters of this world. These unwitting computers can be used economically or militarily against our country at some critical time. Protecting these home computers is a valid concern for the government.

Cross Domain Browsing

Often when working on classified data there is a need to access public documents on the Web. It is critical that those public documents not carry any potentially dangerous code. Remote Web rendering provides high assurance that all malicious code has been purged from the re-written Web page or other internet resource.

Challenges

AJAX

Internet browsing is rapidly changing. No longer are static web pages the norm. AJAX code, which brings in content "asynchronously" (without the page being refreshed), has made the web page more like a traditional software application. AJAX presents one of the biggest challenges to our concept of re-writing of web pages from scratch. AJAX stands for Asynchronous JavaScript and XML. The XML in its name is somewhat of a misnomer, but JavaScript is an integral part of the AJAX technology. New content can be delivered at any time after the initial page loads based on the JavaScript commands that identify when and where that content is to be delivered. Often the new content is delivered upon some user initiated event. For example in the Google Maps web page (I should call it an application) new maps are delivered to the browser in response to the user using the mouse to scroll the maps in any direction. Similarly, on this and other AJAX enabled pages, new content will be delivered in response to on-click, on-page-load, on-mouseover, on-keystroke etc. This allows the page to constantly respond to the actions of the user just as if the application entirely resided on the client machine.

For the rendering server to duplicate this type of AJAX function the rendering server will need to first identify the JavaScript events that are contained in the web page. It will then need to create controls on the new page that will pass on these events to the rendering server, who, in turn, will pass the appropriate data/event on to the document source. Once new content is received at the rendering server it can be rendered or possibly just passed on in purely HTML format for display in at a certain location in the page.

JavaScript

Note that while we are not passing through original JavaScript to the client browser the rendering server will be generating plenty of its own JavaScript that will be sent to the client. Also it will not be passing on original AJAX code to the client browser but will rely heavily on its own AJAX for the interchange between the rendering server and the client browser.

Web 2.0 and beyond

Web 2.0 is mostly due to advances on the server side rather than in the client browser. User built sites rely on user input and interaction. This involves a lot of user inputs being entered into a database for others to view and or edit. When presented in the browser this Web 2.0 content is displayed the same as any other web page. Certainly these “2.0” sites utilize AJAX but not in any different way than other sites. Cloud based functions, on the other hand, are accessed through a web based application and are usually highly AJAX dependent. Each click event typically calls for more or different data from the cloud, as does each keystroke or series of keystrokes. Interaction between the user and the server is pretty much constant throughout the user’s session. To keep this interaction seamless the server must reduce the size of their transmissions to be as small as possible. Integration of these functions into the recreated page will rely on capture of all user actions and constant refresh of certain portions of the page.

Internet plug-ins and Flash

This content provides further complications to the web browser function. Much of the content in Flash could possibly be replaced with HTML 5 features. Whether or not the new HTML 5 specification will introduce vulnerabilities is yet to be determined. Plug-ins present their own difficulties. They can still be installed in the browser but they will be operating on a different set of page data. RWR is not aimed at trying to protect the browser from malicious code in any plug-in. Our only aim is to continue to provide as much functionality of the original web page as possible.

Selectable Content

Text in a web page can usually be copied and posted into another document. In the initial version of our rendering server there is no text to select. The client just sees an image of the text. The challenge in a future version of the rendering server is to provide an avenue for the user to be able to copy the original page text. This could be done by providing selectable see through text over the top of the image of the text.

Design Details

Document Object Model (DOM) as the Transformation Template

The method by which we rewrite the web pages is through creating a screenshot of the page and overlaying the screen shot with features of the page as identified by the Document Object Model (DOM). Simply stated the DOM is a convention for representing the structure and elements in a web page. The JavaScript language in a web page gains many of its capabilities through manipulating the DOM. Through JavaScript we can query the DOM and identify all elements in the web page. In addition, their location on the page can be determined so the elements can be placed over top of the screenshot image and in most cases the user cannot tell that the page has been re-written.

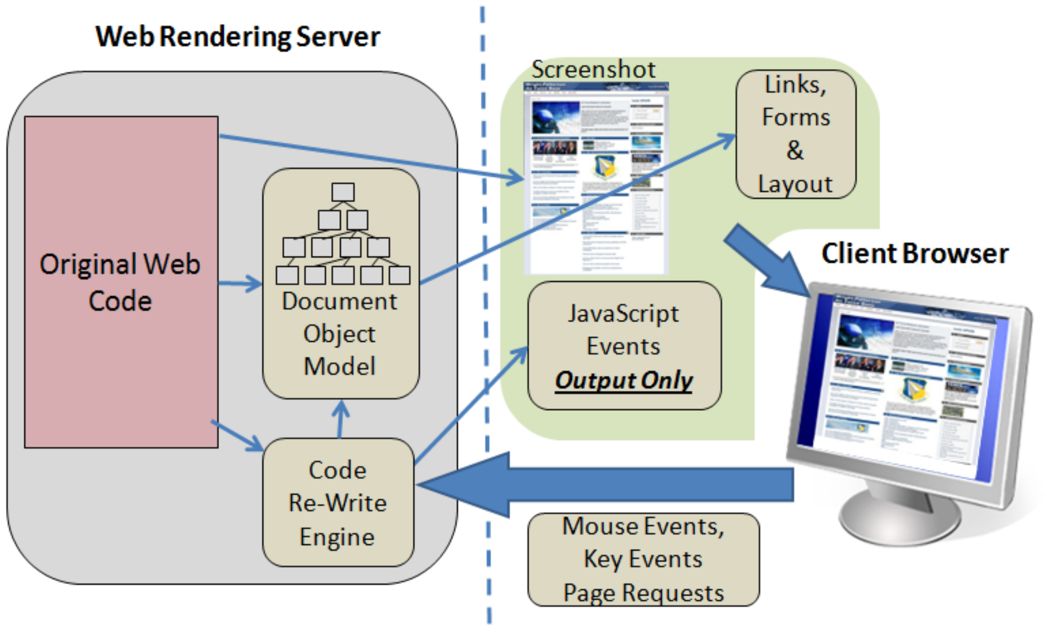


Figure 1 Remote Web Rendering Overview

When rendering the page, the browser also creates the DOM that enumerates elements of the web page. Data contained in the DOM is then used to create the new code that is transferred to the client browser along with the screenshot on which these elements are overlaid. This is an important point: we use the browser's understanding of the elements of the page (as enumerated in the DOM) to create the new code. We are not trying to parse code and then re-create it. If we were parsing code then we could be easily tricked by obfuscation and other tricks of the trade of malware authors. On the contrary we are letting the browser on the rendering server tell us what elements are on the page. If the browser understands that something is a link then we will pass it on as a link. The same is done for other elements that can be extracted from the DOM.

JavaScript that operates in the original web page often makes changes to the DOM. These changes need to be captured by the rendering server and passed on to the client in some form. Often this will change the location of the links in the document. This can affect the re-written page depending on the timing of those DOM changes, i.e. whether it is before or after the DOM has been queried to find the placement of the links. The DOM structure differs from one browser to another but that does not have an effect on our page re-writing method since we are only working with the DOM on the rendering server and we limit that browser to a set type and configuration.

Image Maps as a Page Platform

Early efforts in this research pointed to the possibility of using image maps as the platform for re-writing the web pages but, until the details were worked out, it was uncertain if this was the best course of action. Since this time much work has been done using the image map concept and it is clear that this format works well for re-writing most web pages. The implementation of image map based pages is simple – take a screenshot of the original web page and use that snapshot as a background on which to place links and other page features. The process gets considerably more complicated when you start accounting for flash, JavaScript, AJAX, cookies etc.

For those who are not familiar with image maps they are a standard but somewhat passé Web page format. They allow links and other features to be overlaid over an image by defining hotspots within the image for that link or feature. Since image maps are somewhat ignored in recent years there is little information on the web about their capabilities and how they differ from standard html formats. For instance in the image maps it is customary to define a hotspot in the image using the “area” tag. One difference with this is that the area tag does not support most standard Cascading Style Sheet (CSS) elements. Thus adding a simple hover effect for the area hotspots requires significantly more code.

Even with the differences between image maps and standard html formats we have found that images maps do support the types of content that we need to add when re-writing the web pages. Image maps can support overlays of links, forms, video, layers and html content. They support AJAX data push and pull, and can include JavaScript code just like any other web format. One element that image maps do not support is the dynamic elements that grow and shrink the web page as they are opened and closed. Should these elements be required the rendering server would have to show the element in some other way such as an additional layer that can be closed on command or after a certain time.

Background Image Management

Currently two screenshots are used in the initial rendering of a page. The first screenshot shows just the visible portion of the page as initially seem in the browser window. This is followed a few seconds later by a screenshot of the entire page. Using the two screenshots instead of one allowed a preliminary page to be displayed to the client while the server was given more time to load the entire page and generate a complete screenshot.

An adaptation of this technique would be to screenshot each major portion of the DOM separately. In a way, this would be a little like a thin client solution, where updates were constantly being supplied to the client as each portion of the page changed. It would be an optimization problem to determine at what level in the DOM hierarchy to individually render DOM elements and how often to render those elements. This should be evaluated at a later time.

- Screenshots would be supplied that only showed content in a particular DOM portion. Code for the elements in that DOM portion would have to be supplied with element positioning that is relative to the position of the container rather than being positioned absolutely from the top corner on the screen as it is in the current prototype code.

Separating Top Level Pages from Sub Pages

Determining what internal pages to render and what not to render is not straightforward. More complicated web pages often contain multiple sub pages mostly using internal frames (iframes). One news site we evaluated had three sub pages contained within it and one of those sub pages was duplicated eight separate times – resulting in a total of 11 sub pages on one page. The user scripts, through which we accomplish the rewriting, when injected into the incoming pages cannot always identify their parent. Also, the page that the client requests is often not the page that the client gets due to redirects at the page provider. Only top level pages should be logged in the rendering server database and all sub pages will be included as part of that top level page.

Server Security

Security of the server used for remote rendering is a significant concern. Should a VM on that machine get compromised, or worse yet, the whole machine get compromised, the attacker could have a direct channel to feed malware to one or more clients. The remote rendering methods described herein will have to be proven to be able to address the threats against the rendering box and its software.

In some cases the attacker could be a user (client) of the rendering server. In this capacity they could direct the rendering server to retrieve and render web resources that have been specially crafted to attack the rendering server. This method is similar to what is done at the pwn2own competitions held at CanSecWest. In these competitions hackers were to compromise a fully patched browser or operating system/browser combo. To do so the hackers would typically direct the machine to be compromised at web pages the hackers had built for that hacking competition to exploit that particular OS/browser configuration.

During demonstrations of the remote rendering concept several people observed that we are transferring the risk involved in this process from the client machine to the rendering machine. So, they asked, how does this make us safer? There are several reasons why the rendering server is not just transference of risk with no benefit. They include:

- The rendering server is hosted on a Linux-based operating system (OS). The OS is much more secure than the typical client operating system and will inherently block malicious code. In the pwn2own competitions mentioned above, the Ubuntu Linux operating system was never compromised.
- Each rendering server will provide service for many. This will make patching for client systems less critical while the patching on the rendering server remains critical but can be accomplished easier.
- While Remote Rendering technology does not rely on filters to protect the client from malicious code, it can use the filtering technology for its own protection. It works to the server's advantage to remove known malware from the incoming web page data before it renders the original code. Much of this filtering technology is heavy weight and not appropriate for the installation on multiple client machines. Thus the code that is incoming into the rendering machine is first checked for malicious code before it is rendered. The offending code can then be removed from the file before rendering the page. If the situation warrants the page could also be rejected and a message sent to the client about the presence of malware.
- Virtualization of the rendering server provides a clean slate OS for each browsing session. Should the server OS get infected (doubtful) or a rogue document get uploaded to server file system the system can automatically refresh (and purge the system compromise) at set intervals or for each new client session.
- It is possible that the Remote Rendering server could be hosted in the cloud. In the most secure example of this the user could start up a

new instantiation of the rendering server when they want to access the Web. The server cloud instantiation can then be closed.

In addition to the possibility of being located in the cloud, the rendering server is a good fit for organization gateways, proxy servers and, if we can solve a couple issues, it could be hosted directly on the Web as a portal from which users can surf securely.

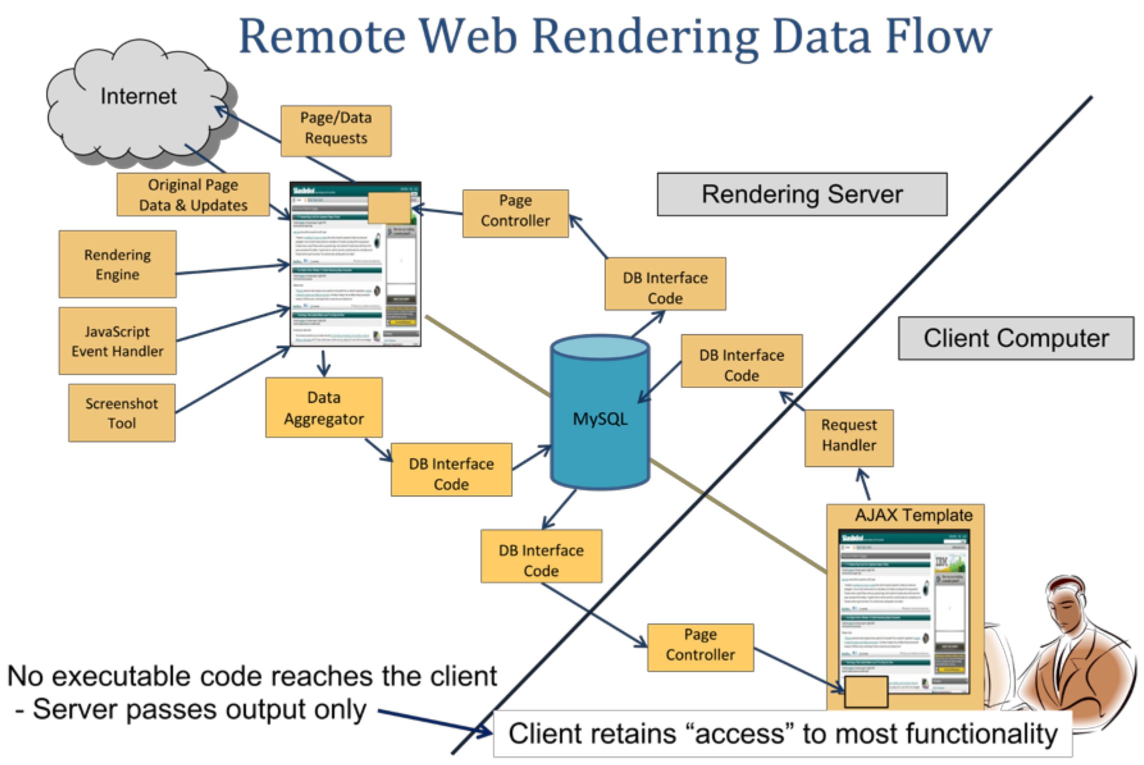


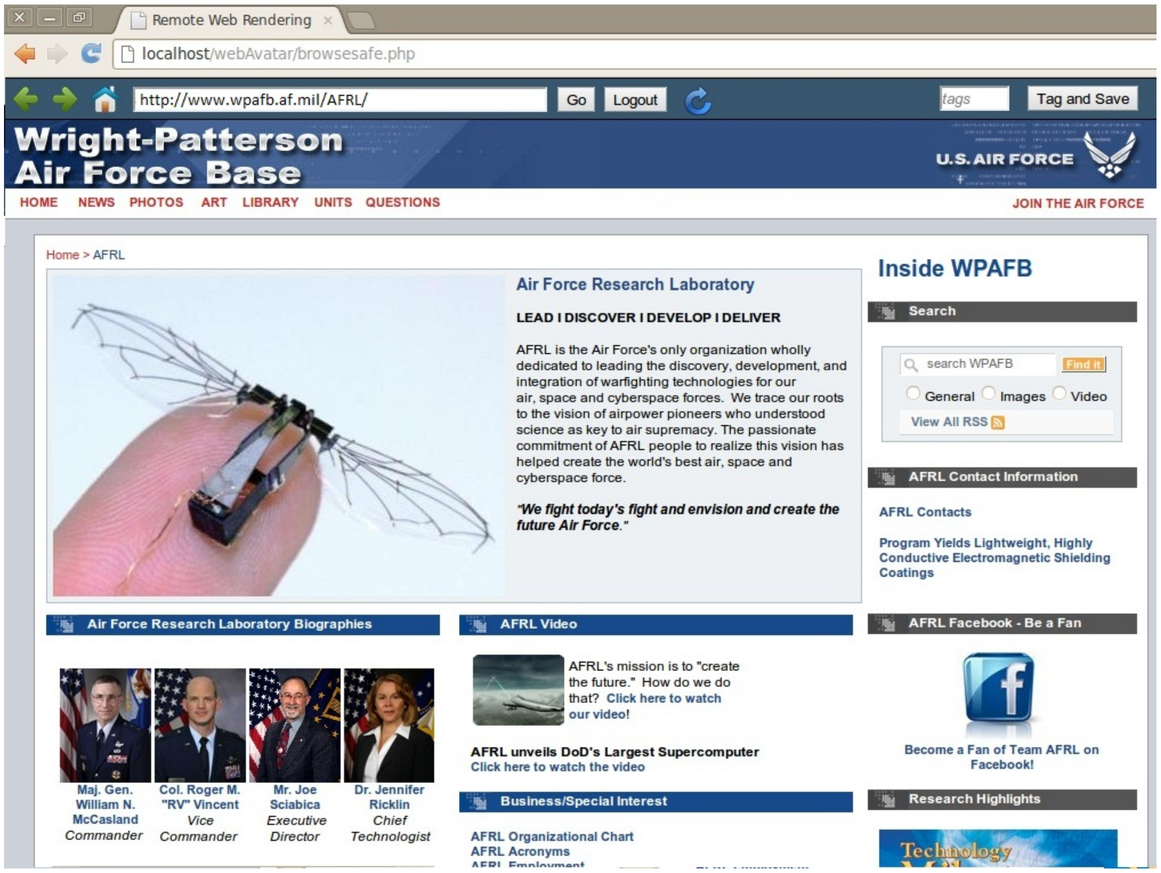
Figure 2 Remote Web Rendering Data Flow

RWR Demonstration Vehicle

Re-writing Web Pages: For demonstration purposes we are using a full up Firefox installation running in Ubuntu with Apache/PHP/MySQL (referred to as the "rendering server"). The client requires no special hardware configuration nor even any additional software installed. The client accesses the remote rendering capability through a normal browser. (Google Chrome is the preferred browser for demonstration purposes but there is no reason why this could not work with other browsers).

Accessing the Remote Web Rendering capability can be done by going to the URL for the rendering server. The rendering page then presents the client with a browser in a browser concept. Figure 3 shows a re-written version of the AFRL homepage page.

Browser level controls such as the address bar, forward and back buttons, history and favorites are all presented via the browser controls presented in the top bar of the page. All interactions between the client and the rendering server can be protected via a secure connection thus making the rendering server a secure way to access the internet from an insecure location.


Remote Web Rendering

localhost/webAvatar/browsesafe.php

http://www.wpafb.af.mil/AFRL/ Go Logout tags Tag and Save

Wright-Patterson Air Force Base

U.S. AIR FORCE

HOME NEWS PHOTOS ART LIBRARY UNITS QUESTIONS JOIN THE AIR FORCE

Home > AFRL

Air Force Research Laboratory

LEAD I DISCOVER I DEVELOP I DELIVER

AFRL is the Air Force's only organization wholly dedicated to leading the discovery, development, and integration of warfighting technologies for our air, space and cyberspace forces. We trace our roots to the vision of airpower pioneers who understood science as key to air supremacy. The passionate commitment of AFRL people to realize this vision has helped create the world's best air, space and cyberspace force.

"We fight today's fight and envision and create the future Air Force."

Inside WPAFB

Search

search WPAFB Find it

General Images Video

View All RSS

AFRL Contact Information

AFRL Contacts

Program Yields Lightweight, Highly Conductive Electromagnetic Shielding Coatings

Air Force Research Laboratory Biographies

Maj. Gen. William N. McCasland Commander

Col. Roger M. "RV" Vincent Vice Commander

Mr. Joe Sciabica Executive Director

Dr. Jennifer Ricklin Chief Technologist

AFRL Video

AFRL's mission is to "create the future." How do we do that? Click here to watch our video!

AFRL unveils DoD's Largest Supercomputer
Click here to watch the video

AFRL Facebook - Be a Fan

Become a Fan of Team AFRL on Facebook!

Business/Special Interest

AFRL Organizational Chart
AFRL Acronyms

Research Highlights

Technology

Figure 3 Non-Proxy RWR Prototype

On the rendering server we inject code (called the rendering script) into the incoming page. This injected script is referred to as a “User Script”. Since our demonstration vehicle has a full up Firefox installation we use the Greasemonkey add-on for injecting the script. In Google Chrome browser there is an add-on called Tamper Monkey for running user scripts. The injected code does the following five functions (which will be discussed in more detail in another section):

1. Determine which is the top level page and which are the sub pages (iframes etc). While the rendering script is injected into each of the pages (sub pages

and top level page) we do not want to confuse the two and show to the client an in-page advertisement as a separate page.

2. Extract information from the DOM concerning the elements of the page. Individual parameters from the DOM elements are extracted and these elements are reassembled in a template of new html code that will be passed on to the client. Of special importance is the fact that we are not looking for whole HTML tags but rather we retrieve the parameters that define a specific HTML tag. Thus, when the rendering server decides that an element is a link, the inserted script will query the DOM and tell us what the URL for that link is, along with all the other parameters that apply to that link. These parameters can then be escaped so that they can be used only for their intended purpose.
3. Repackage the parameters into a new HTML element. The code will then determine the pixel location of the element so that it can be overlaid over a screenshot thus creating an image map of the page.
4. Call a file, "pagesaver.php" that will take the screenshot of the page.
5. Send data to another file, "newfile.php", that will write the rewritten code for the page to the MySQL database.
6. Watch for new requests from the client using an embedded iframe called "server_receive.php".

At the client end we run code to send new page requests to the server and to receive new pages from the server. All these send and receive functions are accomplished via JavaScript/AJAX so the page that the client is on never actually changes, even though the images that they are viewing, and the underlying code for interacting with that page, are constantly changing.

For RWR users not accessing the rendering server as a proxy the client interface is constructed using a frames based HTML page. Here is an overview of the files that are involved:

1. The "Browse" frame displays the browser in a browser controls including address bar, back and forward buttons, a form for tagging and saving favorites, logout, refresh and home buttons. The browser functions are duplicated is to keep the user from accessing the real browser's controls that, in the absence of a web proxy, will cause users to bypass the web rendering server and go direct to unprotected pages.
2. The "Display" frame contains the code for displaying the recreated page to the client. Several pages are used to accomplish this function:
    a. An embedded iframe, "client_receive.php", is used to continually check for updated content from the rendering server and pass that input on to the display page.

b. Another embedded iframe, “requests.php”, is used to process and record requests that result from clicking on links, submitting forms or accessing the new browser control buttons. These client requests are written to the rendering server database where they are “found” by the rendering engine.
c. The top level page, “template.php”, receives the page changes (and new page loads) from client receive. It then processes the inputs and displays to the user the changes to the screenshot and overlays functional links and forms on the screenshot.

Complications arise in duplicating some standard behaviors of the browser. For example the ability to remember the scroll position of the browser window is very helpful when navigating via the back button. Other examples of items that have been, or need to be, addressed in the rendering server code include:

1. Back button, forward button functionality. When the rendering server accesses a page it writes new code to the database. The back button (or history function) works by retrieving a previous record from the data base. When a link is followed from a previous page a new record is written to the database and the client is forwarded to it. This page is then given a new (sequential) ID. At this point the use of the back button should not take the user to the previous page as identified by the sequential ID number. It should take the user to the page that they were previously accessing.
2. Remembering Vertical Scroll state. When hitting the back button the browser should load the previous page and advance the scroll to the point where the user was when they left the page.
3. Anchor tags identify locations in a page where the user could be advanced directly to that vertical scroll position. The anchor is identified in the URL as a “hash tag” that follows a pound symbol. When a URL with a hash tag is called from another page the rendering server will have to render the entire page and write it to the database for the client to retrieve. The client when retrieving it must also identify the scroll position of the anchor tag that the client should be advanced to. Complications arise when the link with the hash tag is accessed that refers to a spot within the existing document. In this case there is no new page for the server to load, just a new location for the client to scroll to.
4. Identifying web pages vs. other file types (ie PDF, Word Docs etc) is not straight forward. Parsing the URL will often not give the right answer. Since each document type should be treated according to a tailored rule set it is important that we be able to distinguish between file types. IE PHP, ASP etc and other web files are often used to create image files, PDFs Word docs etc just by changing the content type in the code.

5. Dead URLs cause problems with the current instantiation of the rendering server. This is because the rendering script does not load in the locally generated “file not found” page. Since the script did not load then there is no script watching for new requests from the client nor is there a way to send on the failed result to the client. It would be helpful to modify on the server the local browser pages that warn the user of this type of situation to include the rendering script. While this is possible to change the local browser pages, it is not easy nor is it documented well.
6. We have implemented a couple different techniques for determining ahead of time when a URL represents a dead link. The first way that we did this was to retrieve a small portion of the page code to determine if there was anything there. In an effort to speed this up we instead used the CURL extension on the rendering server to test each URL. While this is a more efficient solution the CURL request is often treated as unwanted automation by the host server and sent home with an improper http status code for the page.
7. Since a new page is “loaded” by changing the content on the page it is necessary to also implement a control to automatically scroll back to the top of the page when the page is loaded.
8. Back button functionality in the frame based browser bar is tricky and sometimes inconsistent. Calling functions from other frames is sometimes allowed and sometimes not. For example, the back button that is held in the Browse frame does not always launch the back function that is resident in the Display frame. Different browsers treat this cross frame function call differently. For this reason there is also a right-click menu in the Display frame that will allow a user to call the back and forward functions from within that frame.
9. Newer browsers are now starting to understand the AJAX powered site paradigm. As such, the browser back button often realizes that a new page has been displayed to the user even though the page URL has not changed. Thus the browser back and forward buttons that are supported with the client browser will correctly move the page to a different data set just as the internal functions in the remote web rendering code.
10. Image Caching was a problem until we started making the image name unique by inserting the current time. In cases where the page already might have already been in client memory this causes unnecessary delay but in most other cases the unique name is necessary and beneficial.

Screenshots are accomplished via a plugin that that can capture the current visible portion, the entire page or any defined section such as that contained in a particular <div> tag. Reconstructed HTML code that is created by this process is then written to a MySQL database and grabbed by the client from there.

Re-writing other Web Resources Web pages are certainly not the only internet resources that can contain or transmit malware or covert messages. For several years PDF documents were constantly being identified as a malware attack vector due to vulnerabilities in their construction. Image formats also can be used in ways not apparent to the eye. Re-writing these documents for read access is very simple. Recreating these documents in a form that will allow copying text and editing is more complicated.

Document conversion software called ImageMagick can be used to convert multiple files types into alternate types. Conversion in itself should break most if not all malware. IE the malware written into a pdf file will not still be executable if converted to an image format. Similarly the malware that accompanies a Word document will not likely survive the conversion to a pdf format. While ImageMagick is an excellent tool for this type of conversion it is not the only possibility. Documents could be displayed through Google Docs to make the conversion from their current file type to HTML.

Management of the converted files requires the client code on the rendering system to display multiple single page documents in the place of a single multipage document.

Privacy, OPSEC, Industrial Espionage Security

Government employees are continually trained to not give away indications of their intentions. This emphasis is called Operations Security (OPSEC). OPSEC is particularly important in mission planning but it applies to all areas of Government service. For private industry industrial espionage is a big concern that mimics the government's concern for OPSEC. Unfortunately it is impossible to maintain complete security when browsing or searching the internet. Search queries and page views quickly tell a story about our intentions and research focus. Private advertisers currently aggregate this data so that they can provide more focused advertizing – but is this their only intended use for the data? What about the search providers? They can sort data by person or agency and look for spikes in data that indicate breakthroughs or new directives. It goes without saying that search providers and others could quickly discern the current plans for almost any organization.

Identification of the individual who is accessing the page is easily done through watching the IP address or placing a tracking cookie on the client machine. Even if the IP address is masked through a proxy or the client does not accept non-persistent cookies it is possible for the web site to still track the client through a process other parameters that are unique to that particular client computer/browser setup. For example, web pages can often fingerprint the client browser through evaluating multiple browser settings such as the fonts installed, add-ons installed, the versions of these add-ons, etc. The only way to stop this is at the browser, or in the case of Remote Web Rendering, by substituting a remote browser.

This is where Remote Web Rendering can provide additional privacy that is not available during traditional browsing. When the page is remotely rendered all the “fingerprint” parameters that the web site will see are parameters from the rendering server, not from the client computer.

Since multiple users will be using RWR, and all instantiations of RWR are likely to be on identical virtual machines, it will be impossible for the web site to determine who the client is unless they have logged into the page or given some other type of indication of who they are. The web site will get back information about the rendering server but that information will be identical for every user of the rendering server. It will also be impossible to discern what organization the client is from if the rendering server is shared among multiple organizations.

The rendering server can also cache retrieved pages and search results so that when multiple personnel request the same page within a set time frame there will only be a need for one request to go out to the web site for that data. The implications of this for OPSEC and industrial security are significant. Consider the situation when there is a private announcement of new policy or a new product development within an organization. It is highly likely that there will be an immediate spike in the searches done for the keywords that will identify that product or policy. It will be a simple matter for the search provider to identify those spikes and identify the new hot topic at the organization. Caching of similar pages will reduce or eliminate those search keyword spikes or spikes in page access to other telling locations, thus making client (or organization) intentions much harder to discern.

Results

Functionality: Research to this point has concentrated on proving that web resources can be reliably transformed into innocuous forms without significant loss of functionality in a timely manner. The technology has been tested by rendering over 5000 web pages and other internet resources so far. Most pages rendered acceptably and still functions as the original. The ones that did not render acceptably were those that had interactive content run by JavaScript functions. JavaScript functionality transfer to the client was not part of this initial prototype but since a similar concept has successfully been implemented in the WebShield[xi] project it is safe to assume that it would also work in our prototype. PDF documents were converted to a series of images and an interface for interacting with that series of images was created I the client interface.

Performance: Since the prototype system was not optimized there was no attempt to qualitatively evaluate the performance impact of the complete re-writing of the page. The general performance effects (positive and negative) of this process can be categorized as follows:

- The rendering server will likely be on a much better internet connection than the client's internet connection thus the initial rendering of the page on the server will be significantly quicker than it would be on the client machine.
- Actual code writing time on the rendering server is minimal.
- The code size will be much smaller than the code size of the original page.
- Access to the rendering server should be allowed to bypass other proxies and therefore not incur the proxy delay.
- Screenshots from the re-rendered page will typically be much bigger in size than the images in the original page and are the most significant item in the slowing of the delivery of the new page.
- Phased delivery of screenshots enables much quicker access to the top content on the page. It can also be setup to deliver screenshots in a puzzle piece fashion so that no large image file is delivered at one time.
- Reverse AJAX technology needs to be optimized to minimize delays at both client and server.

Overall the Remote Web Rendering concept proved feasible and useful for many potential applications. The prototype implementation, even in its rudimentary form has proven to be useful in a real world setting.

Conclusions

This report describes work done on the Remote Web Rendering system prototype that completely re-writes incoming web resources before they reach the browser. This rewriting effectively blocks all malware and provides a secure way to view any web resource. The prototype demonstrated that this technology is feasible and should be considered for further development by a qualified government or industry team.

Remote Web Rendering presents a solution to a problem that few are trying to solve. We all access the Web daily so we tend to overlook the danger that surrounds us. The Web will host copious amounts of malware for many years to come since there is no possibility of securing the hundreds of thousands of web sites that can pass on malicious code to us. We must protect ourselves from the attacks that bypass our firewall and come through our browser. While Remote Web Rendering is not envisioned at this time to be the total solution to this problem, at least it can provide a significant reduction in Web based client infections in applications where high security is required.

Instances where RWR like approaches should be considered include high security enclaves where browsing may expose critical computers to infection, field deployed troops, and as an alternate proxy for allowing clients to access unknown resources. In addition RWR will also enable encrypted browsing while accessing public WiFi and privacy protections for increasing Operations Security (OPSEC) or industrial espionage security.

Biography

Mr Born has over 25 years experience in conducting and managing research for the Air Force. During that time he has worked in the Reliability, Maintainability, Artificial Intelligence, Planning and Scheduling, Optical Networking and Cyber Defense. He holds patents for prognostics techniques to non-intrusively detect cable chafing and connector corrosion prior to mission impact and has submitted a patent application for the Remote web Rendering technology. In addition, he designed and did much of the coding on a program management web application that has assisted in the management of approximately $4B in research contracts over an 11 year period. Mr Born's current work is in Web application security and browser security. This focus builds on several years of Web development experience in both the public and private sector.

References

[i] Hacked Antivirus Site Delivers a Virus
http://www.pcworld.com/article/142318/hacked_antivirus_site_delivers_a_virus.html

[ii] Hacked Kaspersky Download Site Directs Users to Fake Antivirus
http://www.eweek.com/c/a/Security/Kasperskys-Download-Site-Hacked-Directs-Users-to-Fake-AntiVirus-336193/

[iii] McAfee's Website Full of Security Holes, Researcher Says
http://www.networkworld.com/news/2011/032811-mcafee-security-holes.html

[iv] Hacker lays Claim to Breaches of Two Security Vendors Websites
http://www.darkreading.com/security/attacks-breaches/213401799/hacker-lays-claim-to-breaches-of-two-security-vendors-websites.html

[v] Major Anti-Virus Sites Hacked! http://forum.intern0t.org/security-news-feeds/979-major-anti-virus-sites-hacked.html

[vi] Sophos Corporation "Security Threat Report Update 07/2008"
http://sophos.com/sophos/docs/eng/papers/sophos-security-report-jul08-srna.pdf

[vii] Born, Frank, "Transformative Rendering of Internet Resources", patent pending application serial number 12/802,458, filed May 13, 2010

[viii] CFIBS Cross Fabric Internet Browsing System, http://www.spi.dod.mil/docs/CFIBS_DS_20100422.pdf

[ix] C. Reis, J. Dunagan, H. Wang, O. Dubrovsky, and S. Esmeir. Browsershield: Vulnerability-Driven Filtering of Dynamic HTML. In Proc. of OSDI, 2006

[x] Zhichun Li, Tang Yi et al; WebShield: Enabling Various Web Defense Techniques without Client Side Modifications, http://www.isoc.org/isoc/conferences/ndss/11/pdf/6_2.pdf

[xi] Zhichun Li, Tang Yi et al; WebShield: Enabling Various Web Defense Techniques without Client Side Modifications, http://www.isoc.org/isoc/conferences/ndss/11/pdf/6_2.pdf

[xiv] Niels Provos et al "The Ghost In The Browser, Analysis Of Web-Based Malware" http://www.usenix.org/events/hotbots07/tech/full_papers/provos/provos.pdf, May 2007.

[xv] Harris, Ray. *JavaScript and DOM scripting*, Mike Murach and Associates Inc, 2009.